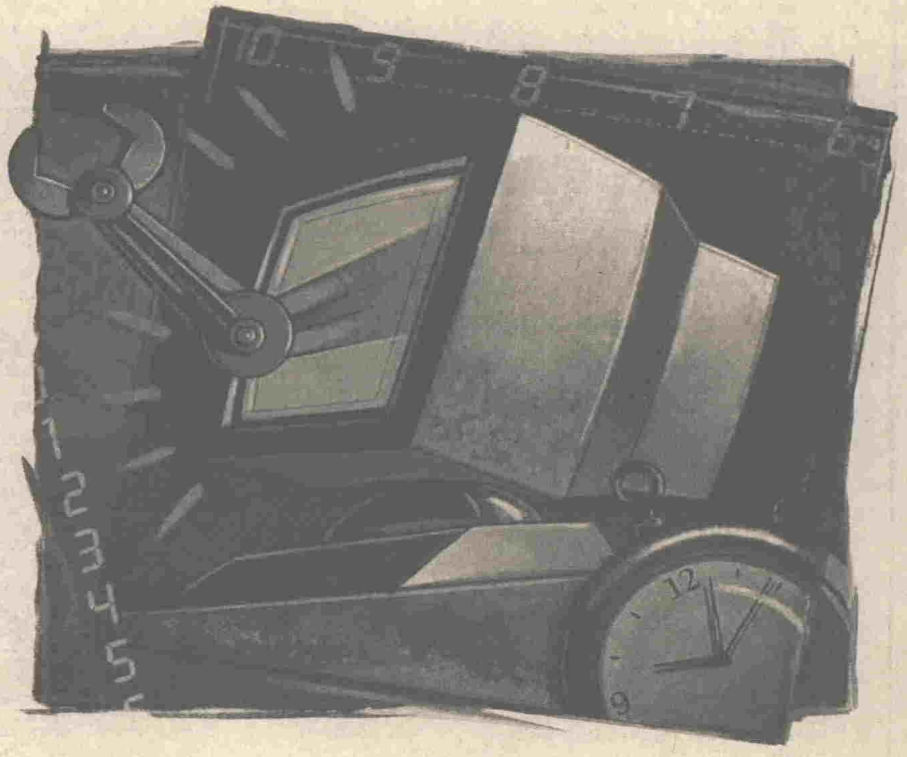
SOFT MAIL

ObjectWindows : หน้าต่างและเม้าส์

ศึกษายุทธวิธีในการเขียนโปรแกรมมิง โดยการสร้างหน้าต่างและการใช้งานกับเม้าส์ภายใน Client area



ผู้อ่านคงจำได้ว่าผู้เขียนเองเคยแนะนำวิธีการเขียนโปรแกรมซึ่งทำงานบนวินโดวส์โดยไม่ต้องใช้ C++ มาแล้ว แต่คราวนี้จะขออนุญาตพูดถึงการเขียนโปรแกรมด้วย C++ สำหรับวิธีการที่จะกล่าวถึงนี้ผู้เขียนจะใช้เครื่องมือที่มีมาในชุดของเทอร์โบ C++ ของบอร์แลนด์ นั่นก็คือ ObjectWindows ซึ่งเป็นคลาสไลบรารีที่จะช่วยในการพัฒนาโปรแกรมได้ง่ายขึ้น เพียงแต่มีเงื่อนไขว่าการเรียกใช้คลาสต่าง ๆ นั้นจะต้องเขียนโปรแกรมด้วยหลักการของ Object Oriented Programming (OOP)

หน้าต่างและเม้าส์

เหตุผลที่ผู้เขียนตั้งชื่อว่าเป็นตอนหน้าต่างและเม้าส์ เนื่องมาจากในบทความนี้ ผู้เขียนจะแสดงวิธีการเขียนโปรแกรม โดยเริ่มจากการสร้างหน้าต่างบานแรกของโปรแกรมและโปรแกรมจะแสดงเหตุการณ์ต่าง ๆ ที่จะเกิดขึ้นเมื่อผู้อ่านใช้งานเม้าส์เช่นเมื่อมีการเลื่อนเม้าส์ โปรแกรมก็จะแสดงค่าโคออร์ดิเนต (X,Y) ภายใน Client area และยังจะแสดงสถานะเมื่อมีการกดหรือปล่อยปุ่มเม้าส์

ผู้อ่านจะสามารถติดตามวิธีเขียนได้เป็นขั้น ๆ โดยผู้เขียนจะอธิบายส่วนประกอบของโปรแกรมในส่วนที่เกี่ยวข้องกับ ObjectWindows ส่วนรายละเอียดอื่น ๆ ขอให้ผู้อ่านอ่านจากบทความต่าง ๆ ที่ลงในหนังสือไมโครเพิ่มเติม

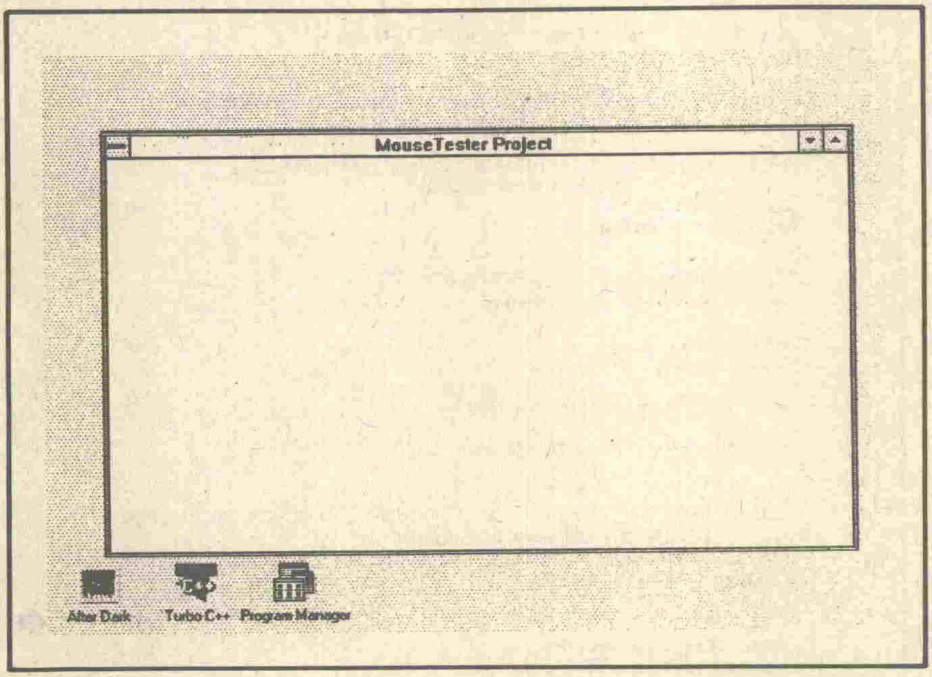
เปิดหน้าต่างบานแรก

การสร้างหน้าต่างสำหรับแอพพลิเคชันนั้นทำได้ง่าย โดยทุกแอพพลิเคชันที่ใช้ ObjectWindows นั้นจะต้องกำหนดคลาสที่ดีไรว์มาจากคลาส TApplication เพื่อจะสืบทอดเอาคุณสมบัติของคลาสมาใช้ คือ สร้าง, แสดง และกำหนดค่าเริ่มต้นต่าง ๆ เช่น โครงสร้างของพอร์ต, โหลดตารางของ Accelerator, ไอคอน, เคอร์เซอร์, จัดการกับเมสเสจของวินโดวส์ จากรูปที่ 1 จะเห็นว่าคลาส TMouseApp เป็นคลาสที่ดีไรว์มาจาก TApplication

ส่วนประกอบหลักของโปรแกรมใด ๆ ก็ตามที่เขียนโดยใช้ ObjectWindows จะต้องประกอบด้วยฟังก์ชัน WinMain เสมอ ซึ่งฟังก์ชันนี้จะเป็นฟังก์ชันแรกที่ถูกเรียกมาทำงานก่อนเมื่อโปรแกรม

```
// ObjectWindows - MOUSETESTER1 PROJECT
// Create MouseTest Window

#include <owl.h>

#define APPNAME "MouseTester Project"

class TMouseApp : public TApplication
{
public:
  TMouseApp(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
  : TApplication (AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
};

void TMouseApp::InitMainWindow()
{
  MainWindow = new TWindow(NULL, Name);
}

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
{
  TMouseApp MouseApp(APPNAME, hInstance, hPrevInstance,
                     lpCmdLine, nCmdShow);
  MouseApp.Run();
  return MouseApp.Status;
}
```

รูปที่ 1 โปรแกรมที่ 1



รูปที่ 2 แสดงหน้าต่างบานแรก เมื่อเรียกโปรแกรมที่ 1 มาใช้งาน

เริ่มทำงาน สำหรับฟังก์ชัน WinMain จะแบ่งออกได้ 3 ส่วนคือ

1. ส่วนที่ใช้สร้างออปเจ็กต์แอพพลิเคชันโดยเรียก Constructor ของคลาสเพื่อกำหนดค่าเริ่มต้นต่าง ๆ ของออปเจ็กต์ (จากรูปที่ 1)

2. ส่วนที่สอง MouseApp.Run( ) เป็นประโยคที่ใช้เริ่มกิจกรรมของตัวแอพพลิเคชัน โดยฟังก์ชัน Run จะเรียกฟังก์ชัน InitApplication และ InitInstance ให้กำหนดค่าเริ่มต้นบางตัวที่จำเป็น จากนั้นจะเรียกฟังก์ชัน InitMainWindow เพื่อสร้างหน้าต่างหลัก ซึ่งผู้อ่านสามารถเปลี่ยนแปลงรายละเอียดได้ (ในโปรแกรมนี้ผู้เขียนไม่ได้แก้ไข) ต่อจากนั้นฟังก์ชัน MessageLoop ก็จะถูกเรียกมาเพื่อทำหน้าที่ตรวจสอบและคอยจัดการกับเมสเสจที่ส่งมาจากวินโดวส์ตลอดเวลาจนกว่าจะปิดแอพพลิเคชัน

3. ส่วนที่จะคืนค่าสถานะต่าง ๆ ของแอพพลิเคชันให้กับวินโดวส์ก่อนที่จะจบการทำงานของแอพพลิเคชัน ปกติมักมีค่าเป็นศูนย์

ถ้าหากผู้อ่านคอมพายล์โปรแกรมในรูปที่ 1 โดยเปิดโปรเจ็กต์ MOUSER1.PRJ ก่อนเมื่อรันโปรแกรมก็จะได้ผลดังรูปที่ 2 ซึ่งเป็นหน้าต่างหลักของแอพพลิเคชันที่ผู้เขียนตั้งชื่อว่า MouseTester Project ซึ่งรายละเอียดและส่วนประกอบของหน้าต่างหลักถูกกำหนดโดยเทอร์โบซีทั้งหมด

บางส่วนของเม้าส์

ก่อนที่จะกล่าวถึงการทำงานบางส่วนของเม้าส์นั้น ผู้เขียนขอกล่าวถึงการสร้างหน้าต่างของแอพพลิเคชันก่อน จากโปรแกรมในรูปที่ 1 จะเห็นว่ารายละเอียด

ต่าง ๆ ของหน้าต่างของแอพพลิเคชันนั้นผู้เขียนไม่ได้เปลี่ยนแปลงเลย แต่ปล่อยให้เป็นตามค่าเริ่มต้นของคลาส TWindow ซึ่งเรียกอยู่ในฟังก์ชัน InitMainWindow

จากโปรแกรมในรูปที่ 3 ผู้เขียนกำหนดคลาส TMouseWindow ขึ้นใหม่ โดยคลาสนี้ดีไรว์มาจากคลาส TWindow เพื่อที่จะสามารถเปลี่ยนแปลงองค์ประกอบและควบคุมการทำงานของหน้าต่างได้เต็มที่ ซึ่งโปรแกรมในรูปที่ 3 นี้จะแสดงการควบคุมการทำงานของเม้าส์ ผู้อ่านคงทราบกันแล้วว่าวินโดวส์จะเป็นผู้ควบคุมการทำงานของเม้าส์ โดยการส่งเมสเสจที่จำเพาะกับเหตุการณ์ต่าง ๆ ที่เกิดจากการทำงานของเม้าส์มาให้แอพพลิเคชันทราบ

การเขียนโปรแกรมจึงจำเป็นต้องเขียนฟังก์ชันย่อยสำหรับเหตุการณ์ต่าง ๆ ที่จะเกิดขึ้นนี้เตรียมไว้ให้วินโดวส์เรียกใช้เมื่อมีเหตุการณ์ดังกล่าวเกิดขึ้น สำหรับการเขียนโปรแกรมด้วย ObjectWindows ทำได้ง่ายมากเพียงแต่ผู้อ่านเขียนเมมเบอร์ฟังก์ชันสำหรับตอบสนองต่อเมสเสจต่าง ๆ ซึ่งเมมเบอร์ฟังก์ชันนี้จะใช้โครงสร้างที่เป็นตารางมาช่วยในการทำงานที่เรียกว่า Dynamic dispatch virtual tables โดยกำหนด Dispatch index เฉพาะสำหรับแต่ละฟังก์ชัน ในที่นี้ได้แก่ WM_LBUTTONDOWN, WM_RBUTTONDOWN ที่ใช้เป็น Dispatch index ของเมมเบอร์ฟังก์ชัน WMLButtonDown, WMRButtonDown จะตอบสนองต่อเหตุการณ์การกดปุ่มซ้ายและขวาของเม้าส์

ถ้าหากผู้อ่านคอมพายล์โปรแกรมในรูปที่ 3 โดยเปิดโปรเจ็กต์ MOUSER2.PRJ ก่อน เมื่อผู้อ่านกดปุ่มเม้าส์

รูปที่ 3 โปรแกรมที่ 2

```
// ObjectWindows - MOUSETESTER2 PROJECT
// Test mouse fuction

#include <owl.h>

#define APPNAME "MouseTester Project"

class TMouseApp : public TApplication
{
public:
  TMouseApp(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
  : TApplication (AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
};

_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
public:
  TMouseWindow(PTWindowsObject AParent, LPSTR ATitle)
  : TWindow(AParent, ATitle) {};
  virtual void WMLButtonDown(RTMessage Msg)
    = [WM_FIRST + WM_LBUTTONDOWN];
  virtual void WMRButtonDown(RTMessage Msg)
    = [WM_FIRST + WM_RBUTTONDOWN];
};

void TMouseWindow::WMLButtonDown(RTMessage)
{
  MessageBox(HWindow, "You have pressed the left mouse button",
    "Message Dispatched", MB_OK);
}

void TMouseWindow::WMRButtonDown(RTMessage)
{
  MessageBox(HWindow, "You have pressed the right mouse button",
    "Message Dispatched", MB_OK);
}

void TMouseApp::InitMainWindow()
{
  MainWindow = new TMouseWindow(NULL, Name);
}

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
           LPSTR lpCmdLine, int nCmdShow)
{
  TMouseApp MouseApp(APPNAME, hInstance, hPrevInstance,
                     lpCmdLine, nCmdShow);
  MouseApp.Run();
  return MouseApp.Status;
}
// ObjectWindows - MOUSETESTER3 PROJECT
// Display some message on window

#include <stdio.h>
#include <owl.h>

#define APPNAME "MouseTester Project"

class TMouseApp : public TApplication
{
public:
  TMouseApp(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
  : TApplication (AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
};

_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
```

รูปที่ 3 (ต่อ)

```
public:
  TMouseWindow(PTWindowsObject AParent, LPSTR ATitle)
  : TWindow(AParent, ATitle) {};
  virtual void Paint(HDC PaintDC, PAINTSTRUCT& PaintInfo);
};

void TMouseWindow::Paint(HDC PaintDC, PAINTSTRUCT&)
{
  TextOut(PaintDC, 20, 20, "Mouse Tester Window Area", 24);
}

void TMouseApp::InitMainWindow()
{
  MainWindow = new TMouseWindow(NULL, Name);
}

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
          LPSTR lpCmdLine, int nCmdShow)
{
  TMouseApp MouseApp(APPNAME, hInstance, hPrevInstance,
                lpCmdLine, nCmdShow);
  MouseApp.Run();
  return MouseApp.Status;
}
```

โปรแกรมจะตอบสนองโดยแสดงหน้าต่างย่อยบอกว่าปุ่มใดถูกกดตามที่ได้เขียนเมมเบอร์ฟังก์ชันที่ตอบสนองการกดปุ่ม

เขียนรายละเอียดบนหน้าต่าง

ผู้อ่านจะเห็นว่าในขณะเริ่มต้นโปรแกรมภายใน Client Area ของหน้าต่างยังว่างเปล่าอยู่ ซึ่งในความเป็นจริงแล้วโปรแกรมทั่วไปภายในหน้าต่างมักต้องมีการตกแต่งรายละเอียดเพื่อที่จะสื่อสารกับผู้ใช้ได้ดีขึ้น สำหรับวิธีการเขียนข้อความใด ๆ ภายในหน้าต่างนั้นทำได้ไม่ยากนักแต่มีวิธีการที่ค่อนข้างจำเพาะ โดยปกติแล้ววินโดวส์จะส่งเมสเสจ WM_PAINT ให้โปรแกรมได้ทราบเมื่อจำเป็นต้องมีการดิสเพลย์

```
#include <stdio.h>
     :
     :
_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
     :
     :
   virtual void Paint(HDC PaintDC, PAINTSTRUCT& PaintInfo);
};
void TMouseWindow::Paint(HDC PaintDC, PAINTSTRUCT&)
{
   TextOut(PaintDC, 20, 20, "Mouse Tester Window Area", 24);
}
```

รูปที่ 4
โปรแกรมที่เพิ่มเติมรายละเอียดในโปรแกรมที่ 2 เพื่อใช้ฟังก์ชัน Paint

```
#include <stdio.h>
     :
     :
_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
     :
     :
   virtual void WMPaint(RTMessage Msg)
     = [WM_FIRST + WM_PAINT];
};
void TMouseWindow::WMPaing(RTMessage)
{
   PAINSTRUCT PS;
   BeginPaint(HWinodw, &PS);
   TextOut(PS.hdc, 20, 20, "Mouse Tester Window Area", 24);
   EndPaint(HWindow, &PS);
}
```

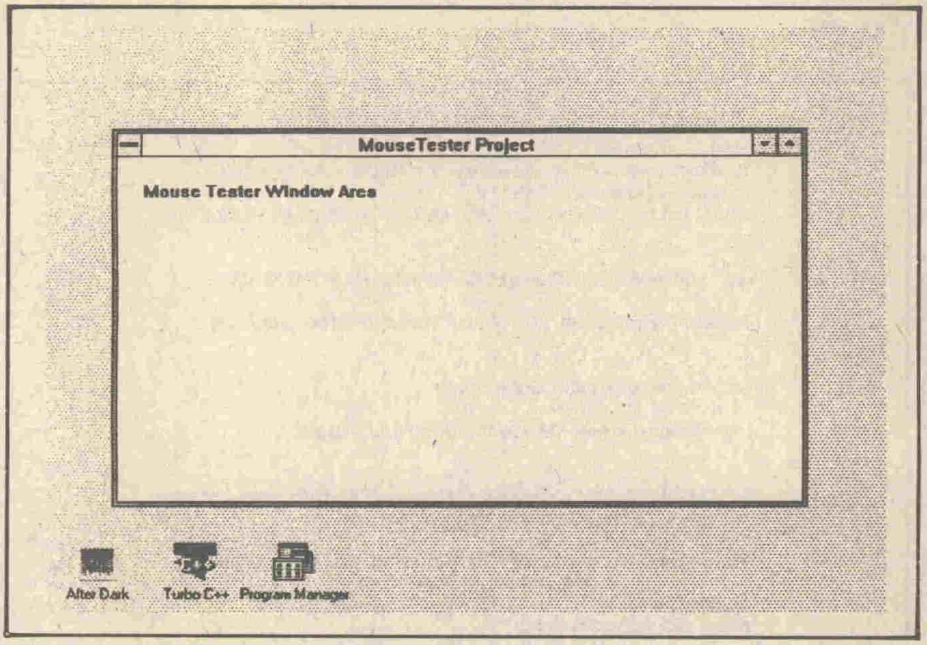
รูปที่ 5
แก้ไขโปรแกรมในรูปที่ 2 เพื่อใช้เมมเบอร์ฟังก์ชัน WMPaint

รายละเอียดของหน้าต่างใหม่ซึ่งผู้อ่านต้องเขียนเมมเบอร์ฟังก์ชัน WMPaint รอไว้ในโปรแกรมด้วย สำหรับ WMPaint แล้วจะเรียกใช้ฟังก์ชัน Paint อีกต่อหนึ่ง การเขียนโปรแกรมสามารถเรียกใช้ฟังก์ชันใดก็ได้ แต่ถ้าใช้ WMPaint ผู้อ่านต้องเรียกฟังก์ชัน BeginPaint ขณะที่เริ่มและ EndPaint จบฟังก์ชันเสมอ

จากโปรแกรมในรูปที่ 3 ให้ผู้อ่านเพิ่มเติมรายละเอียดดังรูปที่ 4

การแก้ไขข้างต้นเป็นการใช้ฟังก์ชัน Paint เพื่อเขียนข้อความลงบนหน้าต่าง คราวนี้ผู้เขียนจะแสดงการใช้เมมเบอร์ฟังก์ชัน WMPaint ให้ผู้อ่านแก้ไขจากโปรแกรมในรูปที่ 3 หรือดูจากไฟล์ MOUSER8.CPP ก็ได้แสดงดังรูปที่ 5

หลังจากที่แก้ไขโปรแกรมหรือจากชุดซอฟต์แวร์ให้ผู้อ่านเปิดโปรเจ็กต์ MOUSER3.PRJ หรือ MOUSER8.PRJ แล้วคอมพายล์และรันโปรแกรมจะได้ผลตามรูปที่ 6 เหมือนกันทั้งวิธีที่เรียกใช้ฟังก์ชัน Paint หรือเมมเบอร์ฟังก์ชัน WMPaint



รูปที่ 6 แสดงผลการเขียนข้อความภายใน Client area ด้วยฟังก์ชัน Paint

```
_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
      :
      :
   TMouseWindow(PTWindowsObject Aparent, LPstr ATitle);
      :
      :
};
TMouseWindow::TMouseWindow(PTWindowsObject Aparent, LPSTR ATitle)
: TWindow(AParent, ATitle)
{
   Attr.X = 50;
   Attr.Y = 50;
   Attr.W = 300;
   Attr.H = 300;
}
```

รูปที่ 7 แก้ไขโปรแกรมที่ 2 เพื่อปรับขนาดของหน้าต่าง

ปรับขนาดหน้าต่าง

ผู้เขียนรู้สึกว่าขนาดของหน้าต่างที่ได้จากรูปที่ 6 นั้นมีขนาดใหญ่เกินไป ดังนั้นจึงทดลองลดขนาดหน้าต่าง โดยปกติแล้วคลาส TWindow จะมีดาต้าเมมเบอร์ชื่อ Attr ซึ่งเป็นโครงสร้าง (structure) ชนิด TWindowAttr ที่เก็บแอตทริบิวต์ของวินโดวส์ขณะที่เริ่มต้นสร้างไว้ ดังนั้นเราจึงสามารถจะแก้ไขแอตทริบิวต์เหล่านี้ได้ขณะที่เรียก Consturctor ของวินโดวส์ที่สร้าง สำหรับแอตทริบิวต์ที่เราสามารถแก้ไขได้มีดังนี้

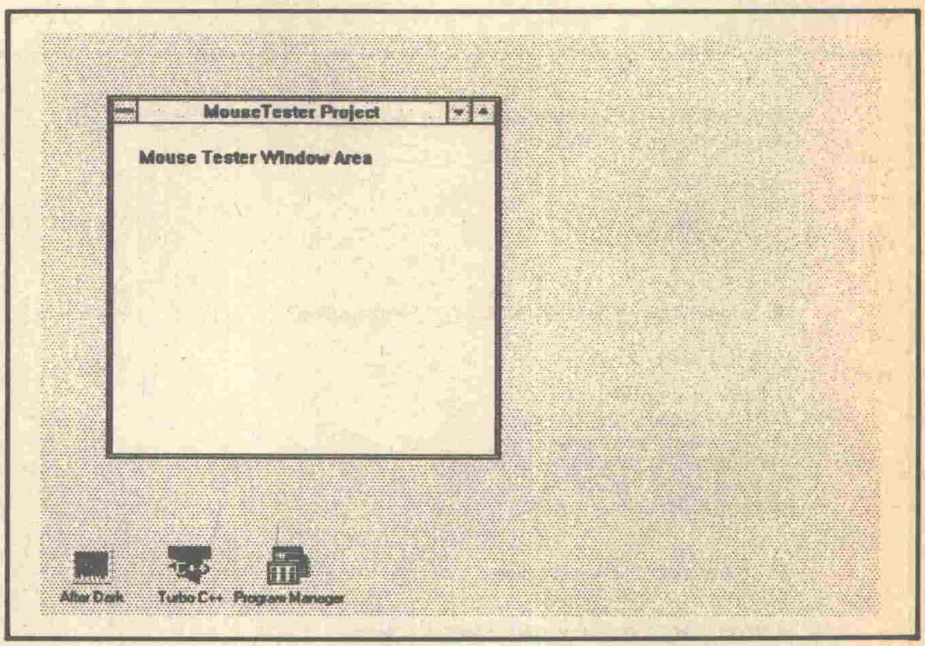
- Style เป็นค่าคงที่ขนาด DWORD ใช้ปรับรูปแบบการแสดงของหน้าต่าง เช่น แบบ Overlap, Pop-up, Border, Caption, Scrollable
- ExStyle เป็นค่าคงที่ขนาด DWORD เช่นกัน
- Menu เป็นข้อมูลชนิด LPSTR ใช้ระบุชื่อเมนูซีซอร์ส
- X เป็นข้อมูลชนิด INT ใช้กำหนดตำแหน่งในแนวนอนของมุมซ้ายบนหน้าต่างที่สร้างใหม่
- Y เป็นข้อมูลชนิด INT ใช้กำหนดตำแหน่งในแนวดิ่งของมุมซ้ายบนหน้าต่างที่สร้างใหม่
- W เป็นข้อมูลชนิด INT ใช้กำหนดความกว้างของหน้าต่างที่สร้าง
- H เป็นข้อมูลชนิด INT ใช้กำหนดความสูงของหน้าต่างที่สร้าง
- Param เป็นข้อมูลชนิด LPSTR

ใช้ส่งข้อมูลให้กับหน้าต่างที่สร้าง

- Id เป็นข้อมูลชนิด INT ใช้กำหนดหมายเลขของหน้าต่างลูกเพื่อใช้ติดต่อกัน

จากโปรแกรมในรูปที่ 3 ผู้เขียนจะแก้ไขให้สามารถปรับขนาดของหน้าต่างได้โดยจะเปลี่ยนแปลงที่ Constructor ของ TMouseWindow ดังรูปที่ 7

เมื่อลองคอมพายล์และรัน MOUSER4.PRJ จะได้ผลดังรูปที่ 8 จะเห็นว่าขนาดของหน้าต่างเมื่อเริ่มต้นจะมีขนาดเล็กลงตามที่กำหนดใหม่ คือ กว้างคูณยาวเท่ากับ 300×300 และมุมซ้ายบนของหน้าต่างก็ย้ายมาอยู่ที่ตำแหน่ง (50,50)



รูปที่ 8 แสดงผลเมื่อปรับขนาดของหน้าต่าง

พาเม้าส์ออกวิ่ง

จากที่ได้กล่าวข้างต้นแล้วว่า การติดต่อของเม้าส์กับโปรแกรมถูกควบคุมโดยการส่งเมสเสจจากวินโดวส์ เมื่อผู้อ่านขยับเม้าส์ให้เคลื่อนที่ วินโดวส์ก็จะส่งเมสเสจที่บอกโปรแกรมว่ามีเหตุการณ์การขยับเม้าส์คือ WM_MOUSEMOVE ดังนั้นจากโครงงานที่วางไว้คือ แสดงตำแหน่ง (X,Y) ที่เม้าส์เคลื่อนที่ผ่าน ก็สามารถทำได้ง่ายคือ ต้องเขียนเมมเบอร์ฟังก์ชัน WMMouseMove ให้ทำหน้าที่เขียนค่าของโคออร์ดิเนตลงบนหน้าต่างนั่นเอง สำหรับค่าโคออร์ดิเนตดังกล่าวจะนำมาได้จากตัวแปร Msg ซึ่งเป็นตัวแปรชนิด RTMessage โดยจะมีข้อมูลชนิด WORD และ LONG คือ W-Param, LParam ซึ่งค่าโคออร์ดิเนต X จะเก็บอยู่ในตัวแปร Msg.LP.Lo และค่า Y จะเก็บอยู่ใน Msg.LP.Hi

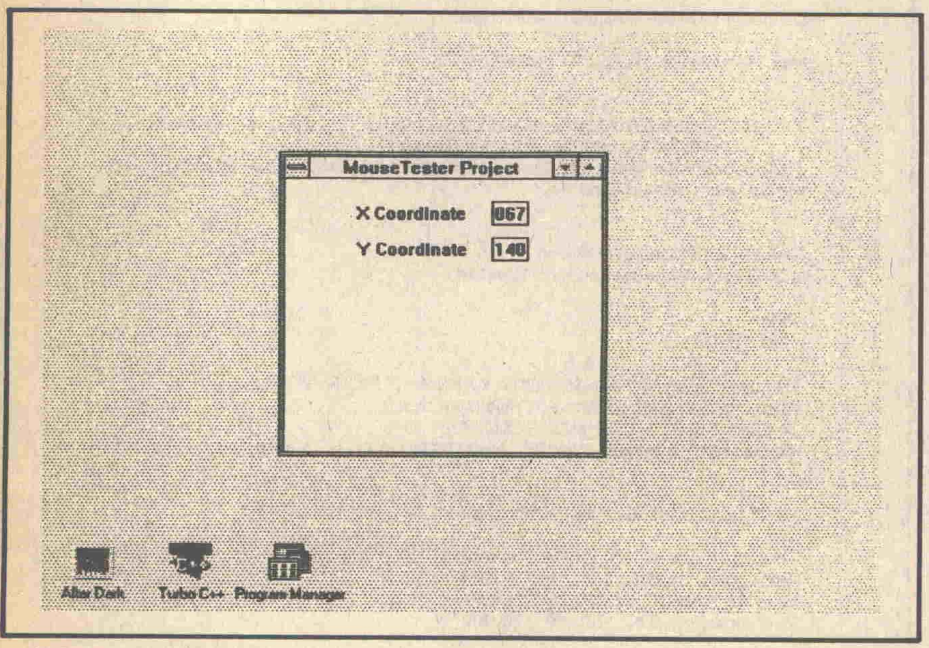
โปรแกรมในรูปที่ 9 ซึ่งแสดงโปรแกรมที่อยู่ในไฟล์ MOUSER5.CPP ที่จะคอยรายงานตำแหน่งที่เม้าส์เคลื่อนที่

รูปที่ 9 โปรแกรมที่ 3 เป็นโปรแกรมที่คอยรายงานการทำงานของเม้าส์ เมื่อเม้าส์มีการเคลื่อนที่

```
// ObjectWindows - MOUSETESTER5 PROJECT
// Running Mouse

#include <stdio.h>
#include <string.h>
#include <owl.h>

#define APPNAME "MouseTester Project"

class TMouseApp : public TApplication
{
public:
  TMouseApp(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
  : TApplication (AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
};

_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
public:
  HDC TheDC;

  TMouseWindow(PTWindowsObject AParent, LPSTR ATitle);
  virtual void WMMouseMove(RTMessage Msg)
    = [WM_FIRST + WM_MOUSEMOVE];
  virtual void Paint(HDC PaintDC, PAINTSTRUCT& PaintInfo);
};

void TMouseWindow::Paint(HDC PaintDC, PAINTSTRUCT&)
{
  TextOut(PaintDC, 55, 20, "X Coordinate", 12);
  Rectangle(PaintDC, 158, 18, 186, 38);
  TextOut(PaintDC, 55, 50, "Y Coordinate", 12);
  Rectangle(PaintDC, 158, 48, 186, 68);
}
```

รูปที่ 9 (ต่อ)

```
TMouseWindow::TMouseWindow(PTWindowsObject AParent, LPSTR ATitle)
: TWindow(AParent, ATitle)
{
  Attr.X = 180;
  Attr.Y = 100;
  Attr.W = 250;
  Attr.H = 250;
}

void TMouseWindow::WMMouseMove(RTMessage Msg)
{
  char X[4], Y[4];

  sprintf(X, "%03d", Msg.LP.Lo);
  sprintf(Y, "%03d", Msg.LP.Hi);
  TheDC = GetDC(HWindow);
  TextOut(TheDC, 160, 20, X, strlen(X));
  TextOut(TheDC, 160, 50, Y, strlen(Y));
  ReleaseDC(HWindow, TheDC);
}

void TMouseApp::InitMainWindow()

{
  MainWindow = new TMouseWindow(NULL, Name);
}

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
          LPSTR lpCmdLine, int nCmdShow)
{
  TMouseApp MouseApp(APPNAME, hInstance, hPrevInstance,
                lpCmdLine, nCmdShow);
  MouseApp.Run();
  return MouseApp.Status;
}
```



รูปที่ 10 แสดงการทำงานของโปรแกรมที่ 3 เมื่อมีการขยับเม้าส์

ขอให้ผู้อ่านลองคอมพายล์ไฟล์ MOUSER 5.PRJ แล้วรันดูก็จะได้ผลดังรูปที่ 10

เพิ่มไอคอนให้โปรแกรม

ผู้เขียนได้กล่าวถึงแอตทริบิวต์ต่าง ๆ ของวินโดวส์ในหัวข้อปรับขนาดหน้าต่าง คือ ดาต้าเมมเบอร์ Attr แอตทริบิวต์เหล่านี้จัดเป็น Creation Attribute ซึ่งเราจะกำหนดได้ในขณะสร้างวินโดวส์ นอกจากแอตทริบิวต์กลุ่มนี้แล้วยังมีแอตทริบิวต์อื่นอีก ได้แก่ เม้าส์เคอร์เซอร์, ไอคอน, สีแบคกราวนด์, แอตทริบิวต์เหล่านี้จัดเป็นวินโดวส์คลาส ก่อนที่จะใช้งานผู้อ่านจะต้องลงทะเบียนไว้ก่อน จึงเรียกว่า Registration Attribute

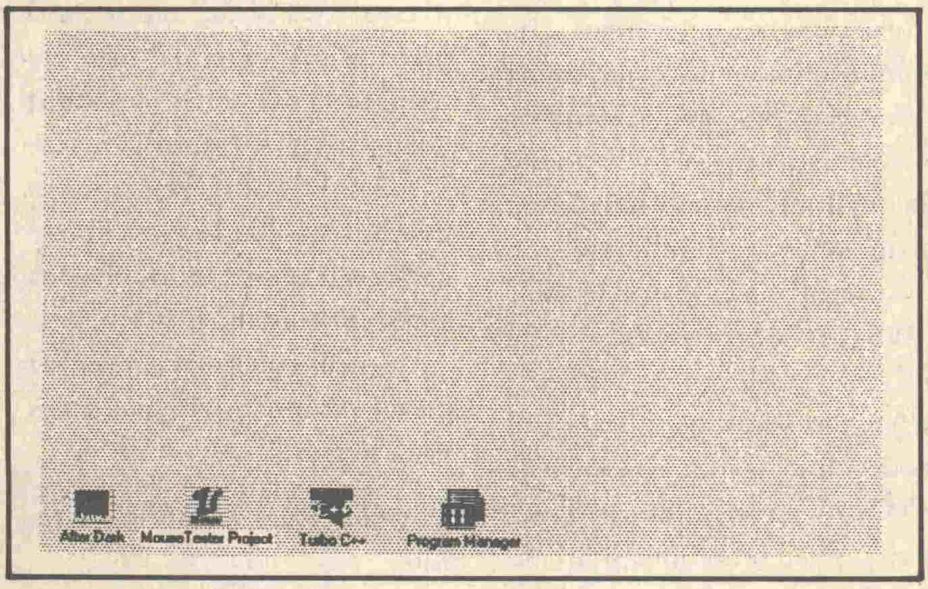
วิธีการแก้ไขแอตทริบิวต์แบบที่สองนี้ทำได้ โดยผู้อ่านต้องแก้ไขที่เมมเบอร์ฟังก์ชัน GetWindowClass และ GetClassName โดยฟังก์ชัน GetWindowClass จะระบุชนิดของแอตทริบิวต์ที่ต้องการแก้ไข ส่วนฟังก์ชัน GetClassName ใช้ลงทะเบียนชื่อวินโดวส์คลาสของแอตทริบิวต์ที่กำหนดใหม่ ขอให้ผู้อ่านแก้โปรแกรมในรูปที่ 9 ให้เป็นดังรูปที่ 11

หลังจากที่แก้ไขแล้วคอมพายล์และรันดูเมื่อผู้อ่านลองย่อโปรแกรมลงเป็นไอคอนจะได้ผลดังรูปที่ 12 สำหรับรูปไอคอนที่ได้นี้ผู้อ่านจะต้องใช้ Resource Workshop ซึ่งเป็นเครื่องมือที่ใช้สร้างไอคอน, เมนู, เคอร์เซอร์, อื่น ๆ ที่มีมาพร้อมกันในชุดของเทอร์โบซีนี้ ผู้เขียนได้ใช้ Resource Workshop สร้างไอคอนดังกล่าวขึ้นมาแล้วตั้งชื่อว่า ICON_1 ซึ่งใช้อ้างถึงในโปรแกรม สำหรับวิธีการใช้งาน Resource Workshop ขออนุญาตยังไม่กล่าวถึงในตอนนี้

ก่อนจะจบหัวข้อเพิ่มไอคอน ผู้-

```
_CLASSDEF(TMouseWindow)
class TMouseWindow : public TWindow
{
     :
     :
   virtual LPSTR GetClassName();
   virtual void GetWindowClass(WNDCLASS&);
     :
     :
};
LPSTR TMouseWindow::GetClassName()
{
   return APPNAME;
}
void TMouseWindow::GetWindowClass(WNDCLASS& AWndClass)
{
   TWindow::GetWindowClass(AWndClass);
   AWndClass.hIcon = LoadIcon(GetApplication()->hInstance, "ICON_1"
}
```

รูปที่ 11
เป็นการแก้ไขโปรแกรมที่ 3
เพื่อแก้ไขแอตทริบิวต์



รูปที่ 12 แสดงลักษณะไอคอนที่สร้างขึ้น

เขียนขอกล่าวถึง Registration Attribute อีกบางส่วน คือแอตทริบิวต์กลุ่มนี้ประกอบด้วย

- Class Style ใช้กำหนด Style ที่จะใช้วาดวินโดวส์ใหม่ ซึ่งจะทำได้ 2 วิธี คือ CS_HREDRAW กำหนดให้วาดใหม่เมื่อมีการเปลี่ยนแปลงขนาดของวินโดวส์ในแนวนอน และ CS_VREDRAW ใช้กำหนดให้วาดใหม่เมื่อมีการเปลี่ยนแปลงขนาดของวินโดวส์ในแนวดิ่ง วิธีการผู้อ่านต้องกำหนดผ่านฟิลด์ Style เช่น

```
AWndClass.style = CS_HREDRAW
: CS VREDRAW;
```

- Icon ใช้กำหนดไอคอนที่ใช้แสดงเมื่อโปรแกรมถูกย่อขนาดลง โดยกำหนดผ่านฟิลด์ hIcon

- Cursor ใช้กำหนดเคอร์เซอร์ของโปรแกรมเพื่อการใช้งานที่ต่างกัน โดยมีเคอร์เซอร์มาตรฐานเช่น IDC_ARROW, IDC_IBEAM, IDC_WAIT, IDC_CROSS มาให้เลือกใช้หรือผู้อ่านอาจวาดขึ้นเองก็ได้ ตัวอย่างเช่น

```
AWndClass.hCursor = LoadCursor (NULL,
IDC_IBEAM);
```

- Background ใช้กำหนดสีของแบคกราวนด์ผ่านฟิลด์ hbrBackground

- Default Menu ใช้กำหนดชื่อของเมนูที่สร้างจาก Resource Workshop เพื่อนำมาใช้ในโปรแกรม โดยกำหนดผ่านฟิลด์ lpszMenuName

ก่อนจบตอน

ในบทความนี้ผู้อ่านได้แสดงวิธีการเขียนโปรแกรมที่ทำงานบนวินโดวส์โดยใช้ ObjectWindows ซึ่งมีมาในเทอร์โบ C++ ผู้เขียนคิดว่าผู้อ่านคงได้แนวทางเพื่อนำไปใช้เขียนโปรแกรมเองบ้าง สำหรับตัวโปรแกรมนั้นขอแนะนำให้ก๊อบปี้จากซอฟต์เมล์คงจะสะดวกกว่าที่จะพิมพ์เอง นอกจากนั้นยังไม่ต้องสร้างโปรเจ็กต์ไฟล์และรีซอร์สไฟล์เองด้วย